ENSC 894: Communication Networks
Spring 2020

Project Final Report

Title: Performance Analysis of Data Centre Network (DCN) architectures using Virtual Network Embedded Simulator (VNE-Sim)

Project Group - Team-01

Malsawmtluangi, Lucy
301371765
lmalsawm@sfu.ca

Fatima, Tehreem
301387807
tehreemf@sfu.ca

Bhogal, Amandeep Singh
301384468
abhogal@sfu.ca

Abioye, Afolabi David
301387738
afolabi_abioye@sfu.ca

https://tehreemf.wixsite.com/vne-sim

April 19, 2020

Contents

List of Figures

List of Tables

List of Abbreviations

BFS Breadth-First Search

BRITE Boston University Representative Internet Topology Generator

CPU Central Processing Unit

D-Vine Deterministic Virtual Network Embedding

DCN Data Centre Network

FNSS Fast Network Simulation Setup

GLPK GNU Linear Programming Kit

GRC Global Resource Capacity

GSL GNU Scientific Library

MaVEn Monte Carlo Virtual Network Embedding

MaVEn-M Monte Carlo Virtual Network Embedding with Multimedia Flow

MaVEn-S Monte Carlo Virtual Network Embedding with Shortest Path

MCF Multi-Commodity Flow

MCTS Monte Carlo Tree Search

MDP Markov Decision Process

MPI Message Passing Interface

R-Vine Randomized Virtual Network Embedding

SDN Software Defined Network

VLiM Virtual Link Mapping

VNE Virtual Network Embedding

VNE-Sim Virtual Network Embedding Simulator

VNoM Virtual Node Mapping

VNR Virtual Network Request

Abstract

The unprecedented expansion of the Internet has led to the introduction of the massive Data Centres that provides the infrastructure for Cloud Computing [8]. Additionally, Software Defined Network (SDN) has aided prevailing Internet architecture with network virtualization to support Cloud Computing [9]. Virtual Network Embedding (VNE) algorithms are leveraged by researchers for implementing virtualization of Data Centre Network (DCN) as it improves network scalability, resource utilization more importantly lowers the implementation cost in DCN's [10]. Moreover, VNE-Sim enables users to define network elements and emulate the infrastructure for both single and batch request processing approaches with good memory management facility [11]. In this project, we will delve into the details of the VNE-Sim discrete event simulator by modeling the networks in Adevs and generating variety of DCN topologies using Boston University Representative Internet Topology Generator (BRITE)[12] and Fast Network Simulation Setup (FNSS)[13], and analyze the performance of the VNE algorithms on new DCN topology (DCell topology) using Hiberlite[14].

Acknowledgement

We are sincerely grateful to our course instructor Dr. Ljiljana Trajkovic and our teaching assistant Zhida Li for their invaluable guidance, and vital suggestions and support during this project and the course. We have gained knowledge and deeper understanding of the exciting computer networking technologies.

And also, to Dr. Sourosh Haeri who developed VNE-Sim for his Ph.D Thesis under the guidance of Dr. Ljiljana Trajkovic at the Communication Networks Laboratory (CNL) at Simon Fraser University, for letting us utilize and work on the latest trend in technology for our project. We are very thankful to both of them for this opportunity to use VNE-Sim for our project and for the insight that we have acquired from this project.

Chapter 1

Introduction

Virtual Network Embedding (VNE) in simple terms means the process of attending to the requests of a VNE-WORK and formation of its respective virtual networks on a substrate network. These requests are mostly presented by the end users that are interested in utilizing the virtual network which are embedded on a specific service provider's infrastructure. The virtual network which is a principal component of VNE process consist of two entities namely: capacity demands of the Central Processing Unit (CPU) for service provider network nodes and the bandwidth capacity demands of the network link. Some other entities like virtual network topology, security constraints can also be included[15].

Moreover, the failure rate as well as the chances of achieving success in a virtual network is primarily dependent on two strict conditions. They are: capacity demands and the availability of resources on the substrate network.

Considering this, VNE can be divided into two different steps of virtual node embedding and virtual link embedding. The former simply means the process of forming virtual nodes on a substrate network while the latter defines the most suitable pathways between the substrate nodes[9].

Furthermore, since attending to virtual network requests is the main objective to define VNE, in some cases the request may be few while they can also be huge, but the virtual network operator must be able to create an accommodating virtual network embedding system. This accommodating VNE system is mandated to hold the available virtual networks and as well make sure the substrate network is efficiently used.

Hence, the process of achieving such a perfect embedding system requires an algorithm which directs an efficient mapping for the huge requests from the virtual network. This algorithm can be termed as Virtual Network Embedding (VNE) Algorithm and its major responsibility is to deal with the problem of resource allocation for both the virtual nodes and links [11][10].

1.1 Motivation and Goals

Computer networks are a very important factor to be considered in an evolving world of today. This is because the world is continuously shifting from the era of traditional form of networking to the software-controlled mode. In this new era, data centers plays major

role as source point to all user-controlled device like phones and computers and these data centers are interconnected through the different wide area network.

Therefore, this caused emergence of Software Defined Network (SDN). Even though SDN offers a more improved level of output to the end-users and the less maintenance effort to the service providers, it has its own limitation. Several algorithms have been designed to check for the acceptance ratio and some characteristic metrics of SDN through some well known network topologies. In this project, we proposed to analyze a new topology called DCell using VNE-Sim and evaluate how DCell would improve the performance of DCN.

Hence, we got interested in this project to know what better improvement a DCell topology offers over the other pre-existing topologies in terms of comparative analysis of their performance metrics.

The objective of this project are listed as follows:

1. Understanding modelling with VNE-SIM for networks.
2. To generate different DCN topologies using BRITE and FNSS
3. To analyze the performance of VNE algorithms in terms of different performance metrics on DCN topologies using Hiberlite.

1.2 Structure of the Report

The remainder of our report is organized as follows: First, we will provide a background study of SDN, virtualization, DCN architectures, and VNE algorithms. In the next section, we will discuss the methods of operation in simulating network embedding to study performance of DCN architectures. Then, we provide a brief description about Virtual Network Embedding Simulator (VNE-Sim), tools used, our code contribution to VNE-Sim, and simulation scenarios of this project. After explaining our understanding of the VNE-Sim, we show the results, and discuss and compare analysis of the scenarios. Finally, we provide the challenges and how we overcome them in the project, with the possible future work for the project.

Chapter 2

Background Study

2.1 Software Defined Network (SDN) - Virtualization

In this era of globally connected networks, information accessibility is at its widespread adoption. At the same time, it is very hard to manage, implement and reconfigure the policies in this connected network and troubleshooting such a complex network itself is a challenging task. In the present network, the data plane and control plane are bundled together which makes it difficult to manage the network. Software Defined Network (SDN) is a paradigm that breaks the integration between data and control plane by implementing logical centralization of network control plane [2]. This centralization separates the control logic from the underlying routers and switches in such a way that these devices just become simple forwarding devices, thus simplifying the policy enforcement and network reconfiguration [2].

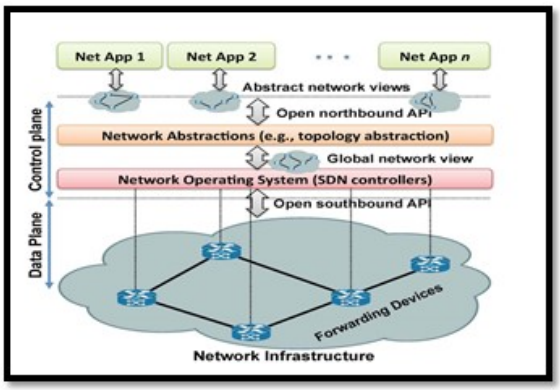


Figure 2.1: SDN Architecture[2]

SDN can be defined as a network with 4 pillars starting with the decoupling of the network and control plane, as the control functionality is removed from the network devices [2]. Secondly, the forwarding decisions are flow based as opposed to destination based to make more unifying behavior of network devices followed by the implementation of the control logic(typically OS) on an external entity called SDN controller that runs on the commodity server to facilitate the programming of forwarding devices [2]. Lastly, the network is made programmable through software applications sitting on top of SDN [2]. Figure 2.1 shows the building blocks, concept and SDN architecture. The control-resource interface between the control and data plane is called southbound interface while the

control-application interface is called northbound interface [16]. Network Virtualization is an abstraction of the underlying infrastructure which provides the networking environment to allow service providers to compose heterogeneous networks which coexist with each other with complete isolation to meet diverse service requirements [16][17]. So, by using the approach of Software Defined Networking, it will be possible to virtualize the data center network with underlying infrastructure. It will result in a network that is simple to manage, and low cost as compared to physical hardware network and efficient to scale according to dynamic requirements [18].

In Network virtualization, we accommodate multiple architectures over the shared substrate network. Virtualization allows coexistence of multiple virtual networks on a shared infrastructure and provides a number of advantages including flexible management, lower implementation cost, higher network scalability, increased resource utilization, and improved energy efficiency. In this project, we will focus server-centric data center network topology called DCell and understand the feasibility of using DCell for network virtualization in DCN by comparing its performance using VNE algorithms such as Vineyard algorithm[19], Global Resource Capacity (GRC) algorithm [10] and Monte Carlo Tree Search (MCTS) algorithm.

2.2 Data Centre Network (DCN) and its Topologies

Data center Network(DCN) is a combination or facility for housing resources (such as computational, storage, and network) interconnected through a communication network. Data Center Network (DCN) topology plays a key role in a data center, as it interconnects all of the data center resources together. DCNs need to be capable enough to be scalable and to connect thousands of servers to handle and manage the increasing demands of Cloud Computing.

Data centers are the fundamental infrastructure for Cloud Computing. Network Virtualization Embedding is one of the key areas of research for future network architecture. According to OSI or TCP/IP model's definition, the DCN operates between layer 2 and layer 3 networks. DCN are connected by either electronic or optical fiber or a mixture of connection types. Moreover, DCN can also be considered as a switch fabric due to the number of network switches which are more in number as compared to the volume of other network components. Commoditized network switches can be adopted to reduce cost of Data Center Networks.

2.2.1 DCN Architectures Classification

DCN architectures are classified into three categories[20]:

1. **Switch-centric Architecture:** In this architecture, all DCN servers are connected by switches.Tree topology is a switch centric topology in which the core layer is responsible for routing and balancing traffic load between the core and the aggregation layer. Aggregation layer provides default gateway redundancy, spanning tree processing, load balancing, and firewall and in the edge layer each switch is connected to two aggregation switches for redundancy. Examples of switch centric topologies are Two Tier, Three-Tier, Fat Tree, F2Tree and Diamond topologies.

(a) **Three-tier DCN:** The Three-tier design of a DCN is based on a hierarchy of switches or routers. It comprises a core layer, the middle tier forms the aggre-gation layer. Access layer switches are connected to the aggregation layer switches and aggregation layer switches are connected to the core layer switches. The core layer is based on switches and routers. Core layer switches are also responsible for connecting the data center to the internet. The aggregate layer switches interconnect multiple access layer switches together. All of the aggre-gate layer switches are connected to each other by core layer switches. Usually the aggregation layer acts as a load balancer.Some of the constraints of this design are oversubscription less than 1:1. This oversubscription defines that all servers talk to each other at full bandwidth of their network interface.At present, three-tier is the commonly used network architecture in data centers. Unfortunately, three-tier architecture is unable to handle the increasing demand of cloud computing. To manage this constraint, data centers need to have mul-tipath routing techniques which require multi-rooted core switches.The servers in the lowest layers are connected directly to one of the edge layer switches. The upper layers of the three-tier DCN are highly oversubscribed which leads to oversubscription, multiplicity of paths, and immensely large rout-ing table entries and increasing lookup latency. The serious drawback of this topol-ogy is excessive cost due to excessive switches in the upper layers.

Moreover, scalability is another important challenge in three-tier DCN. Other challenges faced by the three-tier architecture include cross-sectional band-width, fault tolerance and energy efficiency. The three-tier architecture is based on enterprise-level network devices at the higher layers of topology that are very costly and power consuming[3].

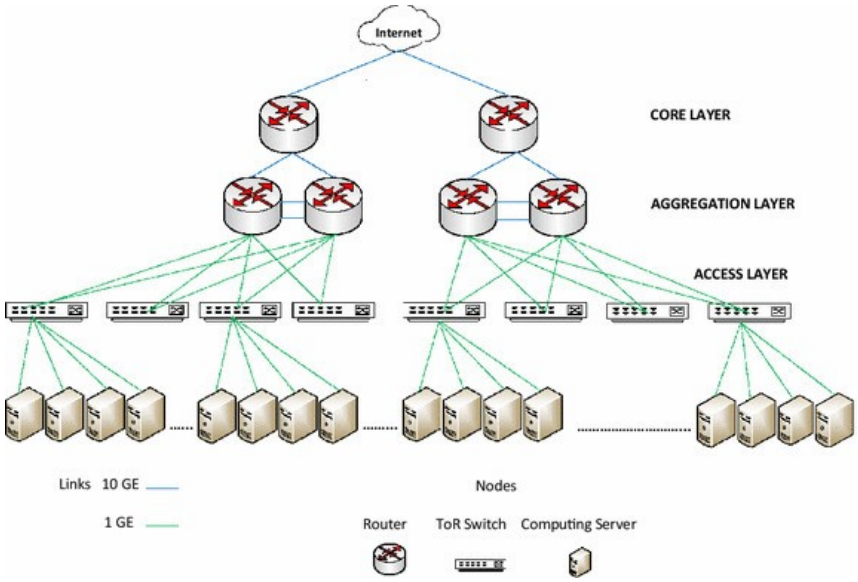


Figure 2.2: Three-tier DCN[3]

(b) **Fat tree DCN:** Fat tree DCN architecture handles the issue of oversubscription and cross section bandwidth problem faced by the three-tier DCN architecture. The fat-tree data center network design incorporates the low cost Ethernet com-modity switches to form a k-ary fat-tree using Clos topology. Fat tree topology also follows hierarchical organization of network switches in access, aggregate, and core layers. However, as compared to three-tier DCN the number of net-work switches is much larger. The fat-tree topology uses identical, commodity

switches in all layers which leads to multiple-times cost reduction.The fat tree topology offers 1:1 oversubscription ratio and employs full bisection bandwidth. The fat tree architecture uses a customized addressing scheme and routing algorithm.

The fat-tree design offers two-level route lookups to assist multi-path routing. "The architecture is composed of k pods, where each pod contains, (k/2)2 servers, k/2 access layer switches, and k/2 aggregate layer switches in the topology. The core layers contain (k/2)2 core switches where each of the core switches is connected to one aggregate layer switch in each of the pods. In order to prevent congestion at a single port due to concentration of traffic to a subnet and to keep the number of prefixes to a limited number, two-level routing tables are used that spread outgoing traffic from IP addresses."[3]

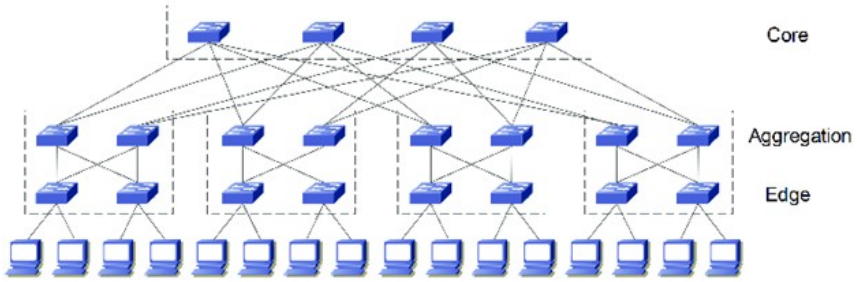


Figure 2.3: Fat tree DCN[4]

2. **Server-Centric Architecture:** In this type of architecture, servers perform computing and packet switching. Bcube, DCell and FiConn are examples of Server-Centric topologies.These recursively defined DCN architectures use the server-centric routing model .

   (a) **DCell:** DCell was introduced in 2008 as a structure which connects multiple servers in a data center. In comparison to Tree Topology, the DCell topology withstands link/node failures and offers a higher bandwidth under heavy load which leads to better performance. In this case, one server is directly connected to many other servers. So, this is a server-centric hybrid DCN architecture . A server in the DCell architecture is equipped with multiple Network Interface Cards (NICs). Due to its recursive and hierarchical characteristics DCell is also able to provide scalability.The DCell structure is composed of basic units called DCells. A cell0 is the building block of DCell topology organized in multiple levels, where a higher-level cell contains multiple lower layer cells. The cell0 contains n servers and one commodity network switch. The DCell uses a hybrid routing and data processing protocol.The network switch is only used to connect the server within a cell0. The DCell routing scheme is used in the DCell architecture to calculate the path from the source to destination node. DCell is a highly scalable architecture.Unlike the conventional routing, the communication with servers in other DCells is performed by servers acting as routers. In addition to scalability, the DCell architecture has a very high structural robustness. However, cross section bandwidth and network latency is a serious issue in DCell DCN architecture[21]

3. **Hybrid Architecture:** This architecture is a combination of switch-centric and server-centric architectures.Hybrid DCN architectures are proposed recently to overcome numerous challenges faced by the legacy electrical-only DCNs. Hybrid DCNs

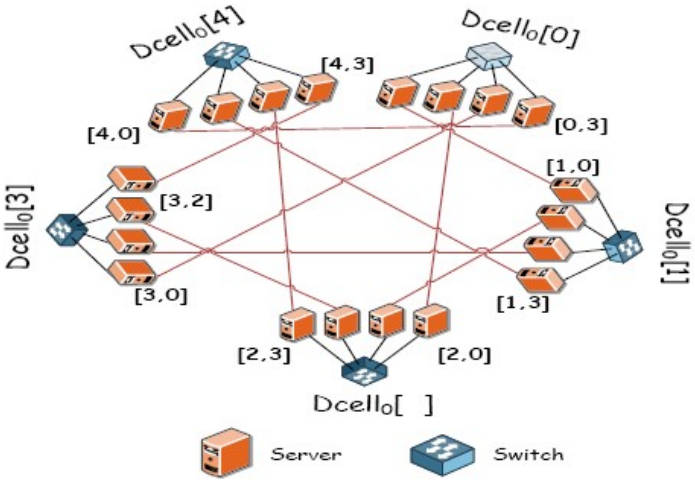


Figure 2.4: DCell Topology [5]

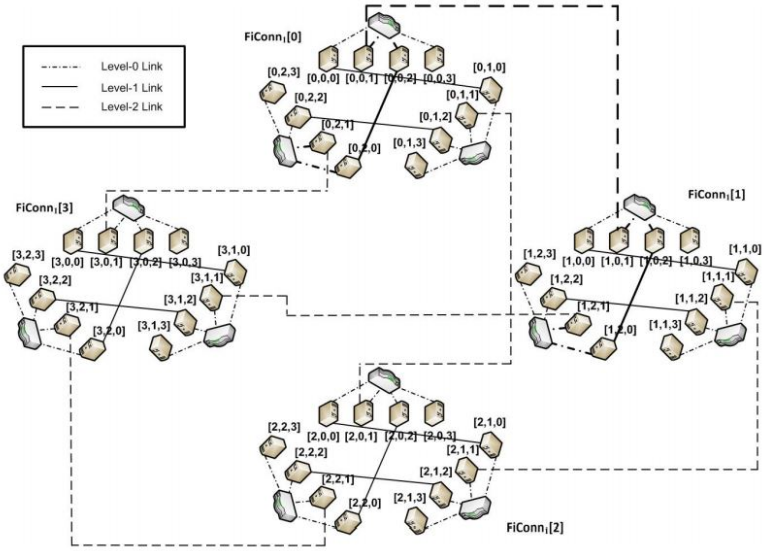
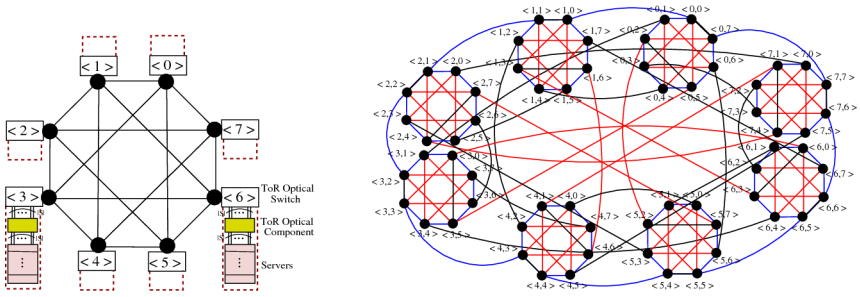


Figure 2.5: FiConn Topology[6]

offer promising opportunities to DCNs. However, hybrids DCNs are facing numerous challenges like interference due to power leaks, and signal reflection in densely populated data centers. Other challenges are such cost, scalability, link setup, switching time, and insertion loss. The wavelength switching time for commercially available optical switches is around 10 to 25ms. Moreover, hybrid networks lack sufficient efficacy for DCNs multi-tenant based mixed and heterogeneous workloads. Hybrid interconnects offer significant network upgrades. The challenges mentioned above are some of the numerous unresolved challenges that are a barrier in adopting hybrid technologies in data centers.

2.2.2 Challenges

Scalability, cross-sectional bandwidth, fault tolerance and energy efficiency are the key challenges to the DCNs. With the emergence of cloud computing, data centers are needed to be scalable to ten ,hundreds and thousands of nodes. Cost performance index is a



(a) Basic Construction Unit, $\Psi(0, \Phi, 8)$

(b) Design of $\Psi(1, \Phi, 8)$ (Two Hamiltonian Cycles are shown in red and blue)

Figure 2.6: Hyscale Topology[7]

critical factor to be taken into account. DCNs are extremely power hungry infrastructures. Recent DCN architectures, such as three-tier DCN offer poor cross-section bandwidth and have very high over-subscription ratio near the root. Fat tree DCN architecture offers 1:1 oversubscription ratio and high cross section bandwidth, but scalability is limited to k=total number of ports in a switch. DCell offers immense scalability, but under heavy network load, its performance is poor. Energy Efficiency is another important parameter.

2.2.3 Performance Analysis of DCNs

Each DCN architecture has a different network topology design with some specific constraints.In this project, we will focus on analyzing the benefits of different data center topologies.The results allow us to indicate the best topology for each scenario. Generally, BCube is more robust to link failures than the other topologies. A detailed analysis of the three-tier, fat tree, and DCell architectures shows that the fat tree DCN delivers high throughput and low latency as compared to three-tier and DCell. DCell topology undergoes very low throughput under high network traffic load and one to many traffic patterns. DCell has the most robust topology while considering switch failures[22].

2.2.4 Energy efficiency of DCNs

The intensifying power consumption has become critically important to modern data centers. Power optimization problems have become serious issues for a general network topology. DCN topologies like fat tree and DCell architectures use commodity network equipment that is inherently energy efficient. Workload consolidation can be a solution.It means that the idle network devices can be shutdown or put into sleep mode for energy saving.

Most of the time, data center traffic is far below the peak value. The idle network devices waste a significant amount of energy. Currently, Data Centre Networks(DCNs) have become significant due to growing demand for information. DCNs should offer high speed data throughput, efficiency and reliability. It has become vital to develop various topologies to organize the operation of DCNs and Traffic Engineering.

Fat Tree topology is one of most common topologies which offers building distributed systems to solve high performance issues. In the Fat Tree topology computing devices are the leaves and nodes are the switches. This topology is called FAT tree, because the connections with other varieties are “thicker.

2.3 VNE Algorithm

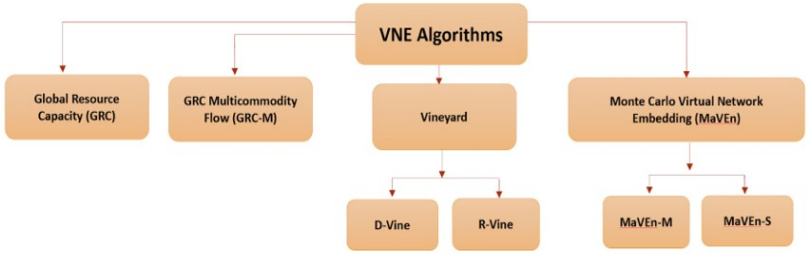


Figure 2.7: VNE Algorithm

In recent times, several works have been done on improving the performance of Virtual Network Embedding (VNE) systems and this therefore led to the following types of algorithms in VNE. They include:

1. **R-Vine and D-Vine Algorithms**
   These two VNE-algorithms were proposed by [19][23] to derive time dependent polynomial algorithms. They arrived at these VNE algorithms by lowering the constraints of the integer program to a linear program and hence they are referred to as Randomized VNE algorithms and Deterministic VNE algorithms.
   In terms of acceptance ratio which is one of the prominent performance metrics of VNE algorithms, it was deduced by [19] that randomization can lead to better performance of VNE algorithm. This in turn means that the R-Vine algorithm offers a better performance with respect to acceptance ratio when compared to its deterministic version of VNE-Algorithm. Work done in [24] also confirmed that D-Vine got a higher revenue of 5% to cost ratio over R-Vine Algorithm.
   Hence, both R-Vine and D-Vine algorithms have distinctive improvements over each other in terms of performance metrics for VNE.

2. **Global Resource Capacity (GRC) Algorithm**
   Taking cognizance of section 1 whereby it was said that VNE can be classified into different steps of Virtual Node Embedding and Virtual Link Embedding. Therefore, the specific VNE algorithm which works with the Virtual Link Mapping is called Global Resource Capacity (GRC).
   With respect to [25], GRC algorithm can be regarded as a "Node Ranking Based Algorithm" that utilizes a modified version of Dijkstra's shortest path algorithm to achieve great performance for Virtual Link Mapping. An improved mode of this algorithm offers more values, and this leads to GRC-M.

3. **GRC-M Algorithm** As mentioned above, GRC-M is an improved version of GRC, and it was proposed by [24]. This algorithm literally means a combination of Global Resource Capacity and Multicommodity Flow Algorithms. Even though according to [25], the GRC algorithm employed the modified version of Dijkstra's algorithm for Virtual Link Mapping, the combination of GRC and MCF which results in GRC-M has been found to improve the performance of VNE. They showed that the acceptance ratio of VNR was further increased by 25% and 10% when compared with the same performance metrics while GRC only is involved.

4. **Monte Carlo Virtual Network Embedding (MaVEn) algorithm** The MaVEn algorithms are newly proposed by Haeri and Trajkovic [11] and they have been approved to offer improved profitability over the well-known ones i.e. D-Vine, R-Vine, GRC and GRC-M algorithms. MaVEn algorithms come in two different types of MaVEn-M and MaVEn-S and they both bring improvements but differ in terms of implementation. While MaVEn-S uses the shortest path algorithm, the MaVEn-M involves MCF for solving virtual link mapping problem.

   According to them [11], MaVEn-M offers a better acceptance ratio over all other compared algorithms but an improved revenue to cost ratio is debatable compared to MaVEn-S based on the effect of traffic load. In terms of node utilization, MaVEn-S has better one over D-Vine, R-Vine, GRC and GRC-M algorithms. Therefore, Maven algorithms both offer distinct improvements with respective limitations, but they are more valuable over R-Vine, D-Vine, GRC and GRC-M algorithms for VNE systems.

Chapter 3

Theory of Operation/Methodology

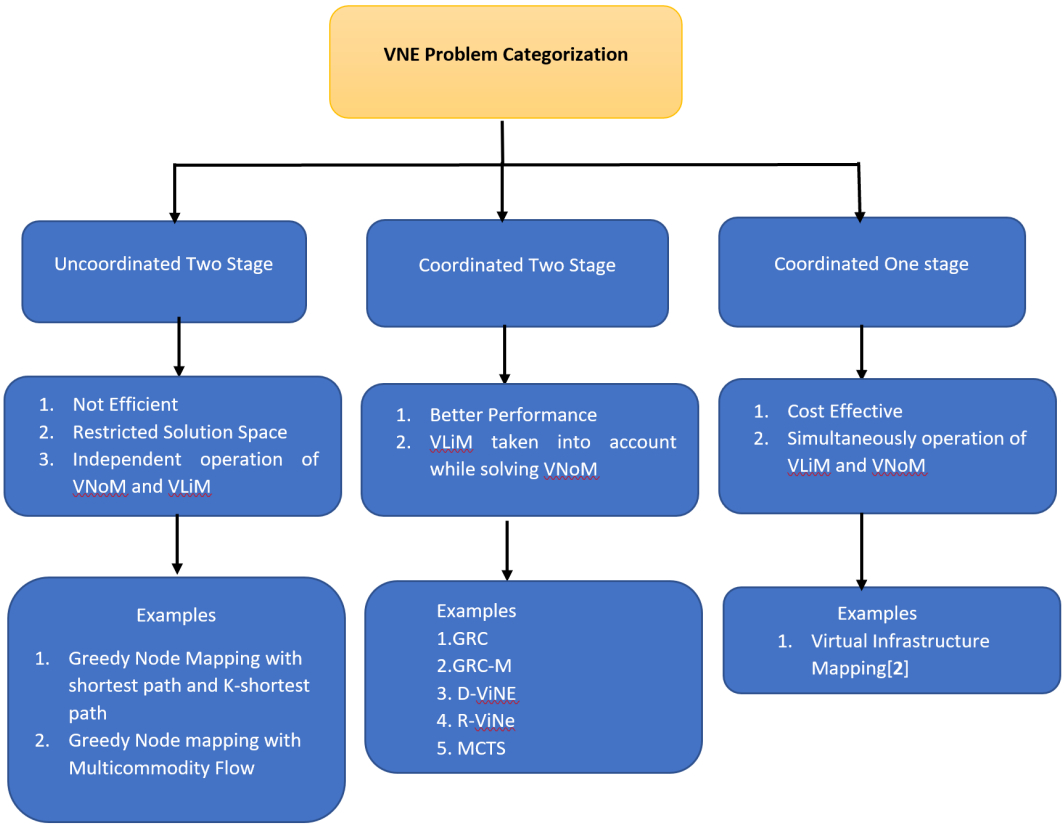


Figure 3.1: Classification of VNE problem [1]

As mentioned in the above diagram, GRC, Vineyard and MCTS are the examples of coordinated two stage algorithms. These algorithms are employed in VNE SIM and we have used these algorithms to compare the performance of the DCN topologies. The GRC algorithm uses node based ranking techniques to solve the VNoM problem. In this algorithm [24], the rank of the node is calculated based upon the bandwidth of the incoming links and the capacity of the substrate nodes. To embed virtual nodes on the substrate node, large to large and small to small mapping schemes are employed [24]. There are two versions of GRC algorithms, the traditional GRC employs BFS to solve for VLiM by determining the shortest path between the nodes using modified Dijkstra's Algorithm, while the Modified algorithm GRC-M uses path splitting based on MCF for

maximising the performance metrics.

The other set of algorithms includes MCTS based MaVEn-M and MaVEn-S which are based on maximizing the rewards for solving VNoM and VLiM problems by employing an agent. To solve for VNoM, it uses Markov Decision Making Process (MDP) in which the agent maps the received VNR to the substrate nodes in finite time. After every successful embedding of an element for a set of virtual nodes, the agent receives reward and the final reward is received after it solves for VLiM. The VLiM problem in MaVEn-M is solved using MCF and MaVEn-S uses the shortest path Breadth First Search(BFS) algorithm.

The Vineyard consists of R-ViNe and D-ViNe algorithms formulates VnoM as mixed integer problem and relax the integer constraints using a rounding based approach to obtain the linear program(LP) for the Mixed Integer Problem(MIP)[26]. To solve the VLiM problem, MCF algorithm is employed to map the virtual links onto the substrate network path. The vineyard algorithms increases the VN request acceptance ratio and subsequently the revenue by making successful use of coordinated node and link mapping technique [19].

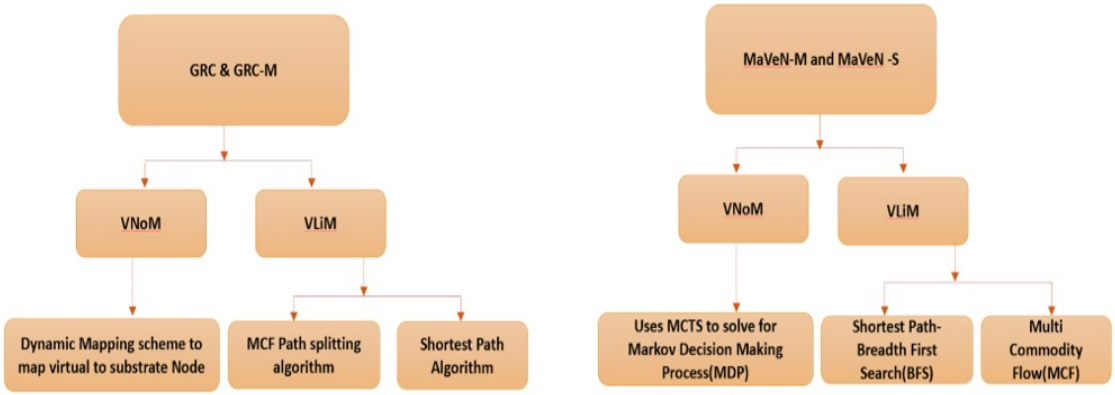


Figure 3.2: Virtual Node Mapping and Virtual Link Mapping [1]

Chapter 4

Virtual Network Embedding Simulator (VNE-Sim)[1]

4.1 High-level Architecture

VNE-Sim is a discrete event simulator written in C++. Several base classes and methods are employed to program the simulation for VNE algorithm, DCN topology, and generate the unprocessed data for discrete events during simulation to a database. The system can be illustrated as several modules as shown in the Appendix A working together in sequence.

1. **Test module**: *src/experiment* contains unit test cases written using Boost Library for Unit Test Framework for various simulation scenarios as follows:
   (a) core-test binary - Use for testing and debugging internal modular functionality with test suite such as NodeTest, LinkTest, NetworkTest, GeneratorTest, VNRequestTest, SubstrateNetworkTest, CoordinateTest, Config, etc.
   (b) experiments-test binary - Use for testing the complete simulation scenario using the different VNE algorithms.
   (c) nfg-test binary - Use for network file generation for substrate network and VNR
   (d) vineyard-test binary - Use for testing and debugging functionality of Vineyard algorithm in particular
2. **Core module**: *src/core* contains classes and interfaces for managing the various template abstract classes for the VNE simulation are maintained in this directory such as, classes for the network components, VNE algorithm, event simulation, and related operation and result collection.
3. **Algorithms module**: The implementation of GRC, Monte Carlo Tree Search (MCTS), Monte Carlo Virtual Network Embedding (MaVEn), and Vineyard (Randomized Virtual Network Embedding (R-Vine) and Deterministic Virtual Network Embedding (D-Vine)) are maintained and organized in their respective directory. VNE-Sim introduces MaVEn algorithms with two variants - Monte Carlo Virtual Network Embedding with Multimedia Flow (MaVEn-M) and Monte Carlo Virtual Network Embedding with Shortest Path (MaVEn-S) and leverages MCTS to solve to solve the problem of Markov Decision Process (MDP) in Virtual Node Mapping

(VNoM). Moreover for Virtual Link Mapping (VLiM), shortest-path algorithms - Breadth-First Search (BFS) and Multi-Commodity Flow (MCF) are used.

4. **Network File Generator (NFG) module**: This module employs BRITE and FNSS libraries to generate network topologies and leverages GNU Scientific Library (GSL) libraries for generating distributions of VNR arrival times, lifetimes, and resource requirements.

5. **External libraries**: Boost[27], GSL[28], Hiberlite[14], Adevs[29], GNU Linear Programming Kit (GLPK)[30], Message Passing Interface (MPI)[31], BRITE[12] and FNSS[13][32] are external libraries that are leverage for VNE simulation.

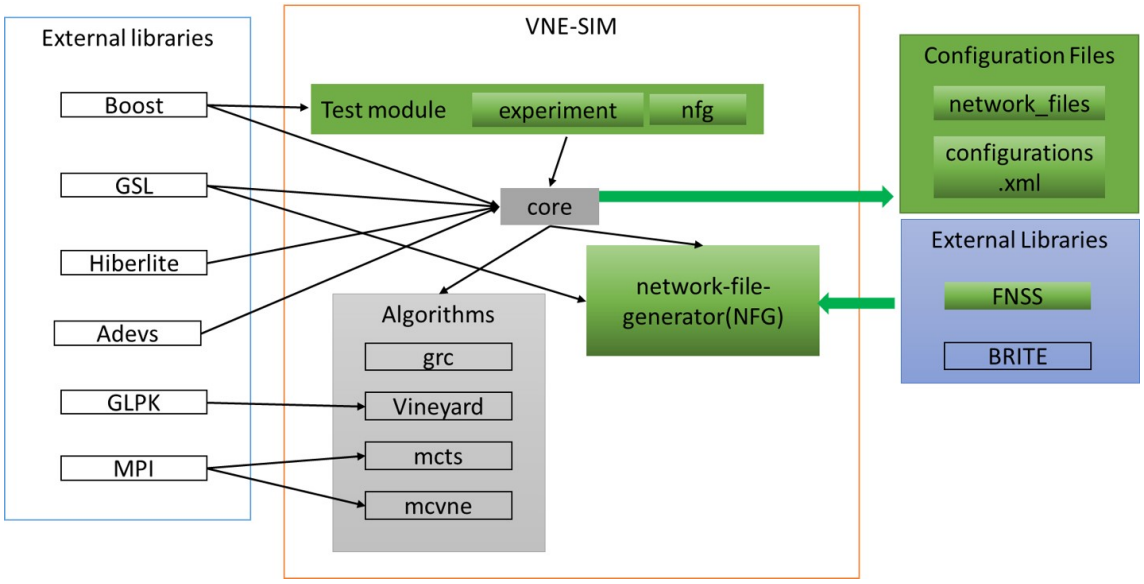


Figure 4.1: High level architecture

4.2 Tools

In this project, we use VNE SIM which is a Virtual Network Embedding discrete event Simulator to evaluate the performance of VNE algorithms. This simulator allows the user to define and enable network elements possessing different resources and at the same time it enables to generate various topologies of Internet Service providers and data center networks for evaluation of VNE algorithms [9]. It works with quite a few external libraries some of which are downloaded and installed using CMake build process while the other libraries are required to be installed before installing this software. It has a well written user guide for selection and implementation of the parameters in order to run the software. In this project we will understand and implement Fast Network Simulation Setup (FNSS)[13][32] and BRITE[12] libraries to simplify the setup of the network and to implement DCell topology in the network.

4.3 Code Enhancement

As we have discussed, VNE-Sim[1] is a software tool which is used for simulating and analysing the performance of the different types of VNE algorithms on DCN topologies.

Table 4.1: Pre-installed libraries compiled with Cmake

Library	Purpose
Boston University Representative Internet Topology Generator (BRITE)[12]	Large scale topology generation for virtual node request as well as substrate network
Fast Network Simulation Setup (FNSS)[13]	Generating DCN topology network scenario for substrate network
Hiberlite[14]	Object relational mapping
Adevs[29]	Modelling the virtual node embedding process as a discrete event system

Table 4.2: Libraries that requires installation

Library	Purpose
Boost File System[27]	Managing File Handling, logging, testing and debugging
GNU Scientific Library (GSL)[28]	Numerical library in C and C++ for generating Random Numbers
GNU Linear Programming Kit (GLPK)[30]	C package for large scale linear programming in VNE algorithms
Message Passing Interface (MPI)[31]	Parallel computing for MCTS(np-hard problems)
SQLite3[33]	Simple Query Language(SQL) Database Engine for handling simulation results
DB Browser[34]	Reading and exporting CSV format for database files compatible with SQLite3

Hence in this project, we enhance VNE-Sim tool to test the performance of a DCN topology which is not included in the VNE-Sim tool and delve in the functional implementation of virtualization in DCN. The areas where we enhanced the code are highlighted as green in Fig.4.1. First, in order to do that we understand the VNE-Sim and the flow of the function calls as elaborated using the Sequence Call in Appendix A. We customized the FNSS package which is leveraged in VNE-Sim uses the substrate network generation. FNSS is distributed as a Python[35] package and is also available as C++ and Java Library[36]. Moreover, FNSS Python package has already support for four DCN topologies: BCube, Two-Tier, Three-Tier and FatTree. Therefore, we added DCell Topology in FNSS which has not been tested before in VNE-Sim. The change process are detailed as follows:

1. **Customizing the FNSS Python Library and implementing DCell Topology**: New DCell topology was added by customizing the FNSS library and the step are as follows: -

   (a) Uninstall the FNSS which was downloaded for building VNE-Sim using the command

   ```
   lucym@ubuntu:~/Downloads/mylib$ pip uninstall fnss
   ```

   (b) Clone the FNSS library and change the name to your choice. (Here, I choose the name as myfnss)

   ```
   lucym@ubuntu:~/Downloads/mylib $ git clone https://github.com/
       fnss/fnss.git myfnss
   ```

(c) Next, change the directory to the newly cloned FNSS package and switch to *root* user.

```
lucym@ubuntu:~/Downloads/mylib$ cd myfnss
lucym@ubuntu:~/Downloads/mylib/myfnss$ su
Password:
```

(d) We need to install FNSS from this folder. The reason behind is that our DCell topology will be added in this library and the changes we made will get reflected when we generate the Substrate network using Python interpreter call in VNE-Sim

```
root@ubuntu:/home/lucym/Downloads/mylib/myfnss# make install
```

(e) Next we add our code for DCell topology, the code additions are listed in Appendix C

```
root@ubuntu:/home/lucym/Downloads/mylib/myfnss# vim fnss/
    topologies/datacenter.py
```

(f) We can test our added Python script as follows and verify using the examples[32] .

```
root@ubuntu:/home/lucym/Downloads/mylib/myfnss# python
Python 2.7.17 (default, Nov  7 2019, 10:07:09)
[GCC 7.4.0] on linux2
Type "help", "copyright", "credits" or "license" for more
    information.
>>> import fnss
>>> topo = fnss.dcell_topology(2, 2)
>>> fnss.write_topology(topo, 'fnss-topo.xml')
>>> exit()
Check the generated topology
root@ubuntu:/home/lucym/Downloads/mylib/myfnss# vim fnss-topo.
    xml
```

2. Next we will list the code enhancement to support DCell topology in VNE-Sim. Code changes are listed in Appendix C.

   (a) First we changes the header file as below to add the structure for DCell and function declaration.
   `src/network-file-generator/fnss-handler.h`

   (b) Next, we add the function definition for the declaration we added in the file below.
   `src/network-file-generator/network-file-generator.cc`

   (c) Finally, we add the calls for the test function in the file below.
   `src/network-file-generator/test/network-file-generator-test.cc`

3. Generate the Substrate network topology

   (a) Compile the changes

   (b) Add the parameter for Substrate network topology in *configurations.xml* file

   (c) Then, run the binary named *nfg* to generate the file

4. Code enhancement for performance testing

   (a) Add the condition to copy the DCNDCell related configuration in setSNNet-Params()
   `src/core/experiment-parameters.cc`

(b) Add the generated file names in `src/experiments/test/experiments-test.cc`

5. Test the performance
   (a) Compile the changes
   (b) Run the experiment binary

4.4 Simulation scenarios

We used the VNE User Guide[37] to setup and install the prerequisite and run the test case. The unit test scenario we have used is called **ARRIVAL_RATE_TESTS**. We use the pre-defined setting in *"reqs-12-1000-nodesMin-3-nodesMax-10-grid-25"* folder. The network files have following configuration: -

- From vnr_brite_param.xml and vnr_param.xml files
  - BRITE RTWaxman algorithm to create connections between the nodes
  - VNR nodes are located on 25x25 Cartesian plane
  - Distance between VNR nodes are 3 and 10. Maximum number of nodes is 3.
  - Each simulation time is set to 50,000-time unit
  - VNR file for arrival distance parameter of 12.5 (traffic of 80 Erlang)
  - Virtual link delay is distributed between 50 and 100
  - Virtual nodes CPU capacity is distributed between 2 and 20
  - Virtual link bandwidth is distributed between 1 and 10
  - Distance to embed VNR nodes is 15 and 25 unit
- From substrate_net_params.xml and substrate_net_generation_algo_params.xml
  - BRITE RTWaxman algorithm to create connections between the nodes
  - VNR nodes are located on 25x25 Cartesian plane
  - Substrate nodes CPU distance, substrate link bandwidth and delay distance parameters are distributed between 50 and 100 units

The code flow is briefly illustrated in Appendix using Sequence Diagram. This unit testcase run all cases the algorithm configuration combinations and the result are stored in SQL DB files (in the order which they were run):

- mcvne_bfs_mcf_reqs-12-1000-nodesMin-3-nodesMax-10-grid-25.db
- mcvne_mcf_mcf_reqs-12-1000-nodesMin-3-nodesMax-10-grid-25.db
- mcvne_bfs_bfs_reqs-12-1000-nodesMin-3-nodesMax-10-grid-25.db
- grc_mcf_reqs-12-1000-nodesMin-3-nodesMax-10-grid-25.db
- grc_bfs_reqs-12-1000-nodesMin-3-nodesMax-10-grid-25.db
- vineyard_deterministic_reqs-12-1000-nodesMin-3-nodesMax-10-grid-25.db
- vineyard_randomized_reqs-12-1000-nodesMin-3-nodesMax-10-grid-25.db

Chapter 5

Result and Analysis

As mentioned above we used VNE-SIM with the required libraries to generate and test the DCell topology in the simulator. We first ran the experiment test case by modifying the files in the simulator. It generated a VNR file for arrival distance parameter of 12.5 with traffic of 80 Erlangs along with an SN file generated for 50 nodes using the BRITE Waxman algorithm. After successfully running initial test cases for validation purposes for GRC, GRC-M , Vineyard and MCTS algorithms, the results are stored in a database directory from which we collect the performance metrics by importing into the DB Browser. By running the test case, we reproduced the results which validates the sufficient implementation of the network modules mainly BRITE and FNSS.

We performed the initial test with the following parameters:

- Virtual Network Request (VNR) and Substrate Network : RT Waxman Graph
- Arrival time: 12.5, Processing time: 50,000 unit time
- Total generated VNR request: 1000

From the results obtained, We deduced that:

- By Comparing the processing time and computation time of the VNE algorithm, the combination of "mcvne bfs mcf" takes the longest time. Logging was enhanced to reduce the simulation time. This will be beneficial once we scale test the performance on realistic DCN configuration.
- The results show that the MaVEn algorithms outperforms Vine and Global Resource Capacity (GRC) VNE algorithms. This was compared using the revenue to cost ratio average as shown below in Table.5.1.
- We can infer from the Table.5.2 that the MCTS and GRC algorithms have better acceptance ratio than Vineyard.
- If the state is EMBED FAIL, the embedding process is unsuccessful, and no revenue is generated because the request is rejected
- Revenue to cost ratio[9] is one of the performance metrics which measure the balance between the revenue generated from embedding VNRs and the cost of allocating

resources. Revenue can be calculated as follows:

$$R(G^v) = w_c \sum_{n^v \in N^v} C(n^v) + w_b \sum_{e^v \in E^v} B(e^v) \tag{5.1}$$

where w_c and w_b are the weights for CPU and bandwidth requirements, and the available CPU capacity of a node n and available bandwidth of a link e are represented by C(n) and B(e).

Cost can be calculated as follows:

$$C(G^v) = \sum_{n^v \in N^v} C(n^v) + w_b \sum_{e^v \in E^v} \sum_{e^s \in E^s} f_{e^s}^{e^v} \tag{5.2}$$

where $f_{e^s}{}^{e^v}$ is the bandwidth of the substrate link e^s that is allocated to the virtual link e^v and C(n) is the the available CPU capacity of a node n

Table 5.1: Algorithm comparison based on Revenue to Cost Ratio - Waxman

Algorithm Combinations	Revenue to cost ratio
GRC-BFS	0.54
GRC-MCF	0.54
MCVNE-BFS	0.68
MCVNE-MCF	0.67
D-Vine MCF	0.49

- Acceptance ratio is another performance metric which measures the ratio of the total number of VNRs accepted (EMBD_SUCCESS) to the total number of VNR that arrived (VNR_ARRIVAL).

Table 5.2: Algorithm comparison based on Acceptance Ratio - Waxman

Algorithm Combinations	Acceptance ratio
GRC-BFS	0.589
GRC-MCF	0.589
MCVNE-BFS	0.593
MCVNE-MCF	0.593
D-Vine MCF	0.581
R-Vine MCF	0.581

For our test on the Dcell topology, we used the following parameters:

- DCell Substrate Network Generation: Using FNSS
- Number of servers on level 0, t = 2
- Number of levels, k = 2
- Virtual Network Request (VNR) Network Generation: Using Brite RTWaxman Graph
- Arrival time: 12, 14, 16, 20, 25, 33, 50, 97

- Processing time: 50,000 unit time
- Request file dataset: 4166, 3571, 3125, 2500, 2000, 1515,1000, 514

For the result using DCell, Revenue to Cost computation in Table 5.3 shows that MCTS algorithm is fairly better than GRC and Vine as well.

Table 5.3: Algorithm comparison based on Revenue to Cost Ratio - Waxman and DCell

Algorithm	Revenue to cost ratio		
	Traffic load - 12 Erlangs	Traffic load - 33 Erlangs	Traffic load - 50 Erlangs
GRC-BFS	0.243	0.243	0.20
GRC-MCF	0.248	0.241	0.23
MCVNE-BFS	0.30	0.297	0.30
MCVNE-MCF	0.304	0.303	0.30
D-Vine MCF	0.246	0.248	0.247
R-Vine MCF	0.246	0.248	0.247

This reflects that DCell is a scalable network structure, and the distributed structure DCell utilizes the resources[21].

For the result using DCell, Acceptance ratio in Table.5.4 reveals that MCTS and Vine algorithms are best. High acceptable ratio is a desirable quality. Currently, DCell shows a low acceptance ratio. The result above shovw the DCell does not perform efficiently when

Table 5.4: Algorithm comparison based on Acceptance Ratio - Waxman and DCell

Algorithm	Acceptance ratio		
	Traffic load - 12 Erlangs	Traffic load - 33 Erlangs	Traffic load - 50 Erlangs
GRC-BFS	0.0549	0.0534	0.055
GRC-MCF	0.0521	0.0594	0.056
MCVNE-BFS	0.0653	0.0627	0.059
MCVNE-MCF	0.066	0.0627	0.059
D-Vine MCF	0.061	0.0592	0.058
R-Vine MCF	0.064	0.0627	0.059

comparing Table.5.2 and Table.5.4. On the other hand, the DCell results contradicts the design motivation of server-centric DCN architectures which is to design topology that can be scaled smoothly as new servers are DCN added. Comparative study by Bilal *et.al.*[38] shows that as DCell uses the inter-link connection between the servers at level 0 which can lead to congestion, packet delay and packet loss in larger networks. When we look into the root-cause of the problem of large embedding failure which causes the low acceptance ratio, we found that the number of link/node to the link/node mapping of the substrate network and virtual network. This means that parameters used for data-points generation for the VNRs need to be tuned further to rectify the problem.

Chapter 6

Challenges and Future Work

Eary from the start, the first struggle with the Virtual Network Embedding Simulator (VNE-Sim) was the long simulation time which took around 3 days for the complete simulation run. We tried various workarounds to deal with this issue and the change that we stick with finally was disabling the Boost logging module. Once we understand the VNE-Sim, the next challenge was with Virtual Network Request (VNR) datapoints generation. Due to the limitations of our system, we could not generate more than a traffic load of 97 Erlangs. After the problem with generating data-points, we faced a problem in embedding Python in VNE-Sim to call the FNSS function to generate the substrate network. The issue was in the backward incompatibility between the Python packages version that was used when VNE-Sim was designed and implemented. However, we overcome this issue with code fixes. As we explained with Fig.4.1, VNE-Sim is a well-designed and organized standalone tool which was used in our project to study VNE operation and to understand the tradeoff between different DCN architecture in terms of the different performance metrics. We were able to successfully implement a DCN topology called DCell and test the performance on various traffic loads. Therefore, the tool is amazing and would only require aesthetic changes such as using newer versions of software, removing all the warnings, introducing added-features to the tool such as topology animation, *etc.* that improve the user-experience of using the tool.

Chapter 7

Conclusion

To map virtual nodes to physical nodes in Virtual Network Embedding Simulator (VNE-Sim), we introduced an efficient strategy, called DCell topology to interconnect servers. Each server is connected to a different level of DCells via its multiple links, but all the servers act equally. High-level DCells are built recursively from many low-level DCells. DCell uses only mini-switches to scale out, and it scales doubly exponentially with the server node degree.

Therefore, a DCell with a small server node degree can support up to several millions of servers without using core switches/routers. On top of its interconnection structure, DCell is also fault-tolerant and its performance is based on computation of shortest-path routing. Moreover, DCell offers much higher network capacity compared with two tier, diamond and the tree-based topologies. Traffic carried by DCell is distributed quite evenly across all links; there is no severe bottleneck. The best application scenario for DCell is large data centers[21].

Expected outcome comprises the development of an algorithm which would offer flexible management, lower implementation cost, higher network scalability, increased resource utilization, feasibility analysis under interference constraints, and improved energy efficiency.In this Project, we compared Vineyard algorithms, GRC, and MaVEn algorithms using two performance metrics acceptance ratio, revenue to cost ratio. Based on simulation results, we analyzed behavior of a DCell Topology. We expect to embed a virtual node to the substrate node which has the minimum sum of the influence distance from already selected substrate nodes and embed virtual links to the shortest influence path. Our main goal is power savings by consolidating network resources in networks and data centers which directed our interest in DCell. However, we were not able to achieve our expected result. Searching for the cause we found that we need to tune the parameter for datapoints generation more correctly. Furthermore, we were able to add to the proof that Monte Carlo Virtual Network Embedding (MaVEn) algorithm seems to work best in all architectures of Data Centre Network (DCN)

Appendix A

Sequence Diagram

To illustrate how the VNE-Sim works when we run the unit test-case, we will use a sequence diagram in Fig.A.1 to show how the modules communicate with other modules briefly.

1. The test scenario starts by setting the path for the VNR and node embedding algorithm – BFS-SP and MCF.
2. Then initialize the instance for the type of VNE-algorithm and corresponding virtual link algorithm for example, MCVNENodeMCFLinkExp and initialize the instance of the substrate network builder.
3. VNE implementation uses co-ordinated two stage algorithms - node mapping algorithm to map the virtual nodes onto substrate nodes and a link mapping algorithm for mapping virtual links onto substrate paths.
4. Three discrete processes are used for modelling VNE process embedding – embedding process, release process and VNR generation process, and an additional observer process to handle the different events.
5. After fetching the dbPath information from configuration.xml the database to created.
6. All the required instances are now initialized, we can run the simulation.
7. We create a new instance for the database to save the completed statistics
8. We destroy the instances of the singleton classes – ConfigManager, IDManager, and RNG to free the resources for the next simulation

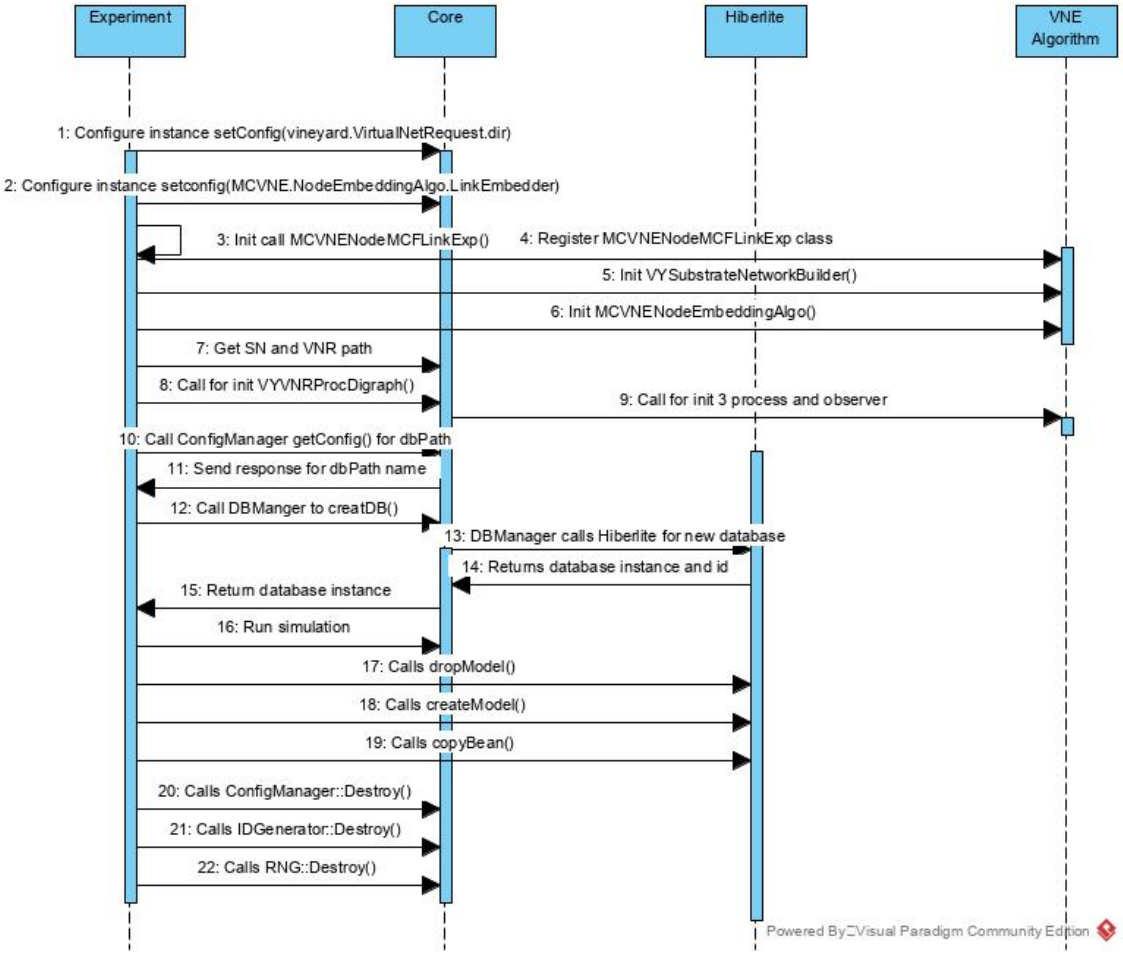


Figure A.1: Brief illustration of a sequence flow

Appendix B

Software for running VNE-Sim

This Appendix is in support of the VNE-Sim User Guide [37] and to illustrate our experiences in the installing VNE-Sim, making it run. The version of software we used in Ubuntu are as follows:

1. Ubuntu 18.04.4 LTS version
2. Cmake version - 3.10.2
3. Boost libraries version - 1.65.1
4. GNU Scientific Library (GSL) version - 2.4
5. GNU Linear Programming Kit (GPLK) version - 4.65
6. pip version - 20.0.2
7. FNSS Python package version - 0.9.0
8. SQLite3 version - 3.22.0
9. DB Browser version - 3.10.1

Appendix C

Code Change Listing

C.1 FNSS Package customization

File name: fnss/topologies/datacenter.py
First, add the new function declaration for DCell topology

```
__all__ = [
'DatacenterTopology',
'two_tier_topology',
'three_tier_topology',
'bcube_topology',
'fat_tree_topology',
'dcell_topology'
]
```

Then, add the function definition

```
def dcell_topology(t, k):
        """
        Return a DCell datacenter topology, as described in [1]_:

        DCell uses a recursively-defined structure to interconnect
        servers. Each server connects to different levels of DCells
        via multiple links. We build high-level DCells recursively
        from many low-level ones, in a way that the low level DCells
        form a fully-connected graph. Due to its structure, DCell
        uses only mini-switches to scale out instead of using
        high-end switches to scale up, and it scales doubly
        exponentially with the server node degree.

        This topology comprises:
        * :math:`g_k = t_k-1 + 1` DCells
        * :math:`t_k = g_k * t_k-1` servers
        * :DCell0 is a special case when g_0 = 1 and t_0 = n, with n
            being the number of servers in a DCell0
        * :Node degree of a server is k+1

        Each node has an attribute type which can either be *switch* or
            *host*
        and an attribute *level* which specifies at what level of the
            Bcube
```

```
    hierarchy it is located.

    Each edge also has the attribute *level*.

    Parameters
    ----------
    k : int
            The level of DCell
    t : int
            The number of host per :math:`DCell_0`

    Returns
    -------
    topology : DatacenterTopology

    References
    ----------
    .. [1] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu,
            "Dcell: a scalable and fault tolerant network structure
            for data centers," in Proceedings of the ACM SIGCOMM
            2008 conference on Data communication, 2008, pp. 75-86
    """

    # Validate input arguments
    if not isinstance(t, int) or not isinstance(k, int):
    raise TypeError('k and n arguments must be of int type')
    if t < 0:
            raise ValueError("Invalid n parameter. It should be >=
                0")
    if k < -1:
            raise ValueError("Invalid k parameter. It should be >=
                -1")

    topo = DatacenterTopology(type='dcell')
    topo.name = "dcell_topology(%d,%d)" % (t, k)

    # Calculate the total number of switches and host from n and k
    totaln_switches = 1
    totaln_host = t

    if (k > -1):
            for _ in range(k):
                    totaln_switches *= totaln_host + 1
                    totaln_host *= totaln_host + 1
                    totaln_switches += totaln_host

    tl = []  # the number of servers in DCell_l
    g = []  # the number of DCell_(l-1)s in DCell_l

    tl.append(t)
    g.append(1)

    for i in range(k):
            g.append(tl[i] + 1)
            tl.append(g[i + 1] * tl[i])

    _host = {}
    _switches = {}
```

```
pref = []

# Create all the switches
for i in range(totaln_switches):
        switch_name = "s" + str(i)
        num = str(pref + [i])
        _switches[num] = switch_name

# Create all the host
for i in range(totaln_host):
        host_name = "h" + str(i)
        num = str(pref + [i])
        _host[num] = host_name

n_host = 0
n_switch0 = tl[k]-1
n_switch = 0

for i in range (tl[k]/t):
        num = str(pref + [n_switch0])
        switch_name0 = _switches[num]
        topo.add_node(switch_name0, type = "switch")
        n_switch0 += 1
        for j in range (t):
                host_id = str(pref + [n_host])
                host_name = _host[host_id]
                topo.add_node(host_name, type="host")
                n_host += 1
                switch_id = str(pref + [n_switch])
                switch_name = _switches[switch_id]
                topo.add_node(switch_name, level=0, type = "
                    switch")
                n_switch += 1
                topo.add_edge(switch_name, host_name)
                topo.add_edge(switch_name0, switch_name)

for m in range(1, k+1):
        num_host = tl[m-1]
        num_dcells = tl[k]/num_host


        for i in range (num_dcells):
                for j in range (i+1, num_dcells):
                        x = (i*num_host+j-1)
                        y = (j*num_host+i)
                        if (x > (tl[k]-1)):
                                a = x - tl[k]
                        else:
                                a = x
                        if (y > (tl[k]-1)):
                                b = y - tl[k]
                        else:
                                b = y

                        id1 = str(pref + [a])
                        id2 = str(pref + [b])
                        s1 = _switches[id1]
                        s2 = _switches[id2]
```

```
                        if topo.has_edge(s1,s2) == True:
                                continue
                        elif topo.has_edge(s2, s1) == True:
                                continue
                        else:
                                topo.add_edge(s1, s2)

return topo
```

C.2 VNE-Sim enhancement

File name: src/network-file-generator/fnss-handler.h

Add a new structure for DCell topology configuration.

```
struct DCNDCell
{
        DCNDCell();
        int t;          /* The number of servers at DCell level
           0 */
        int k;          /* The number of level(s) */
}dcell;
```

Next, add the declaration for API to call FNSS function to generate DCell Topology

```
std::shared_ptr<Network<A, B>> getNetwork_DCNDCell
        (Distribution cpu_dist, double cpu_param1, double
           cpu_param2, double cpu_param3,
        Distribution bw_dist, double bw_param1, double bw_param2
           , double bw_param3,
        Distribution delay_dist, double delay_param1, double
           delay_param2, double delay_param3);
```

Then, add the definition *getNetwork_DCNDCell()*of the function as well,

```
template<typename A, typename B>
        std::shared_ptr<Network<A, B>> FNSSHandler<A,B>::
           getNetwork_DCNDCell
                (Distribution cpu_dist, double cpu_param1,
                   double cpu_param2, double cpu_param3,
                Distribution bw_dist, double bw_param1, double
                   bw_param2, double bw_param3,
                Distribution delay_dist, double delay_param1,
                   double delay_param2, double delay_param3)
{
        std::stringstream pythonScript;
        pythonScript << "import fnss;";
        pythonScript << "topology = " << "fnss.dcell_topology(t=
           " << params.dcell.t << ",k=" << params.dcell.k << "
           );";
        pythonScript << "fnss.write_topology(topology, '.
           datacenter_topology.xml')";
```

```
PyRun_SimpleString(pythonScript.str().c_str());

fnss::Topology t = fnss::Parser::parseTopology(".
    datacenter_topology.xml");
std::set<std::pair <std::string, std::string> > edges =
    t.getAllEdges();
std::set<std::string> nodes = t.getAllNodes();
assert (nodes.size() > 0 && edges.size() > 0);

std::shared_ptr<Network<A, B>> net (new Network<A, B>())
    ;
std::map<std::string, int> fnssNodeIdToVNESimNodeId;

for(set<string>::iterator it = nodes.begin(); it !=
    nodes.end(); it++)
{
        fnss::Node fnssNode = t.getNode(*it);

        std::shared_ptr<A> n = nullptr;
        // If the node is a host create it with a cpu
            capacity
        if (fnssNode.getProperty("type").compare("host")
            == 0)
        {
                numHosts++;
                double node_cpu = RNG::Instance()->
                    sampleDistribution<double,double,
                    double,double>
                (cpu_dist, std::tuple<double,double,
                    double> (cpu_param1, cpu_param2,
                    cpu_param3)) ;
                n.reset (new A (node_cpu, 0, 0));
        }
        else
        {
                numSwitches++;
                n.reset (new A (0,0,0));
        }
        fnssNodeIdToVNESimNodeId[*it] = n->getId();
        net->addNode (n);
}

for(set<pair <string, string> >::iterator it = edges.
    begin(); it != edges.end(); it++)
{
        double link_bw = RNG::Instance()->
            sampleDistribution<double,double,double,
            double>
        (bw_dist, std::tuple<double,double,double> (
            bw_param1, bw_param2, bw_param3));
```

```
                double link_delay = RNG::Instance()->
                    sampleDistribution<double,double,double,
                    double>
                (delay_dist, std::tuple<double,double,double> (
                    delay_param1, delay_param2, delay_param3));

                int nodeFromId = fnssNodeIdToVNESimNodeId[(*it).
                    first];
                int nodeToId = fnssNodeIdToVNESimNodeId[(*it).
                    second];

                std::shared_ptr<B> l (new B (link_bw, link_delay
                    , nodeFromId, nodeToId));
                net->addLink (l);
            }
            this->pt.put ("n_switches", numSwitches);
            this->pt.put ("n_hosts", numHosts);
            this->pt.put ("n_links", net->getNumLinks());
            return net;
}
```

Next, we add the template function for **FNSSHandler**

```
template <typename A, typename B>
        FNSSHandler<A,B>::Parameters::DCNDCell::DCNDCell () :
        t(ConfigManager::Instance()->getConfig<int>("
            NetworkFileGenerator.FNSSHandler.DCNDCell.T")),
        k(ConfigManager::Instance()->getConfig<int>("
            NetworkFileGenerator.FNSSHandler.DCNDCell.K"))
{
}

template<typename A, typename B>
        FNSSHandler<A,B>::Parameters::Parameters () :
        bcube (DCNBCube()),
        twotier (DCNTwoTier()),
        dcell (DCNDCell()),
        fattree (DCNFatTree())
{
}

template <typename A, typename B>
        FNSSHandler<A,B>::FNSSHandler () :
        ExternalLibHandler<A,B> (),
        params(Parameters())
{
        this->pt.put ("DCNBCube.N", params.bcube.n);
        this->pt.put ("DCNBCube.K", params.bcube.k);
        this->pt.put ("DCNFatTree.coreBWMultiplier", params.
            fattree.coreBWMultiplier);
        this->pt.put ("DCNFatTree.K", params.fattree.k);
        this->pt.put ("DCNTwoTier.n_core", params.twotier.n_core
```

```
            );
        this->pt.put ("DCNTwoTier.n_edges", params.twotier.
            n_edges);
        this->pt.put ("DCNTwoTier.n_hosts", params.twotier.
            n_hosts);
        this->pt.put ("DCNTwoTier.coreBWMultiplier", params.
            twotier.coreBWMultiplier);
        this->pt.put ("DCNDCell.T", params.dcell.t);
        this->pt.put ("DCNDCell.K", params.dcell.k);
}
```

File name: src/core/experiment-parameters.h

```
118             ar & HIBERLITE_NVP (sn_dcell_t);
119             ar & HIBERLITE_NVP (sn_dcell_k);

216         int sn_dcell_t;
217         int sn_dcell_k;
```

File name: src/core/experiment-parameters.cc

```
void hibernate(Archive & ar)
127         else if (tt == Topology_Type::DCNDCell)
128         {
129             printf("Log: setSNNetParams()\n");
130             sn_dcn_n_switches = pt.get<int> ("n_switches");
131             sn_dcn_n_hosts = pt.get<int> ("n_hosts");
132             sn_dcn_n_link = pt.get<int>("n_links");
133             sn_dcell_k = pt.get<int>("DCNDCell.K");
134             sn_dcell_t = pt.get<int>("DCNDCell.T");
135         }
```

Bibliography

[1] S. Haeri. (2019) Vne-sim: A virtual network embedding simulator. Accessed: 2020-03-06. [Online]. Available: http://www.sfu.ca/~ljilja/cnl/projects/VNE-Sim/vne-sim-web/index.html

[2] D. Kreutz, F. M. V. Ramos, P. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, p. 63, 2015. [Online]. Available: http://arxiv.org/abs/1406.0440

[3] S. Zafar, A. Bashir, and S. Chaudhry, "On implementation of dctcp on three-tier and fat-tree data center network topologies," *SpringerPlus*, vol. 5, 12 2016.

[4] T. Wang, Z. Su, Y. Xia, and M. Hamdi, "Rethinking the data center networking: Architecture, network protocols, and resource sharing," *IEEE Access*, vol. 2, pp. 1481–1496, 01 2014.

[5] M. Rogers, R. Mugonza, and J. Businge, "Tools, architectures and techniques for monitoring energy efficiency in computer networks: State of the art survey," *International Journal of Computer Applications*, vol. 146, pp. 16–23, 07 2016.

[6] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "Ficonn: Using backup port for server interconnection in data centers," 05 2009, pp. 2276 – 2285.

[7] S. Saha, J. S. Deogun, and L. Xu, "Hyscale: A hybrid optical network based scalable, switch-centric architecture for data centers," *2012 IEEE International Conference on Communications (ICC)*, pp. 2934–2938, 2012.

[8] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi, *Data center networks: Topologies, architectures and fault-tolerance characteristics*. Springer Science & Business Media, 2013.

[9] S. Haeri and L. Trajkovic, "Vne-sim: A virtual network embedding simulator," in *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS'16. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, p. 112–117.

[10] H. Ben Yedder, Q. Ding, U. Zakia, Z. Li, S. Haeri, and L. Trajkovic, "Comparison of virtualization algorithms and topologies for data center networks," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, July 2017, pp. 1–6.

[11] S. Haeri and L. Trajkovic, "Virtual network embedding via monte carlo tree search," *IEEE Transactions on Cybernetics*, vol. PP, pp. 1–12, 02 2017.

[12] A. Medina, A. Lakhina, I. Matta, and J. Byers. (2019) Boston university representative internet topology generator. Accessed: 2020-03-06. [Online]. Available: http://www.cs.bu.edu/brite/

[13] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, ser. SimuTools '13. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, p. 82–91.

[14] P. Korzhyk. (2011) Hiberlite. Accessed: 2020-03-06. [Online]. Available: https://code.google.com/archive/p/hiberlite/

[15] D.-L. Nguyen, H. Byun, N. Kim, and C.-K. Kim, "Towards efficient dynamic virtual network embedding strategy for cloud iot networks," *International Journal of Distributed Sensor Networks*, vol. 14, 01 2018.

[16] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, p. 862–876, Apr. 2010. [Online]. Available: https://doi.org/10.1016/j.comnet.2009.10.017

[17] Q. Duan, N. Ansari, and M. Toy, "Software-defined network virtualization: an architectural framework for integrating sdn and nfv for service provisioning in future networks," *IEEE Network*, vol. 30, no. 5, pp. 10–16, Sep. 2016.

[18] B. Hedlund. (2011) Network virtualization is like a big virtual chassis. Accessed: 2020-03-06. [Online]. Available: http://bradhedlund.com/2011/10/12/network-virtualization-is-like-a-big-virtual-chassis/

[19] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on networking*, vol. 20, no. 1, pp. 206–219, 2011.

[20] H. Qi, M. Shiraz, A. Gani, M. Whaiduzzaman, and S. Khan, "Sierpinski triangle based data center architecture in cloud computing," *The Journal of Supercomputing*, vol. 69, pp. 1–21, 08 2014.

[21] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, 2008, pp. 75–86.

[22] J. Ali, S. Lee, and B.-h. Roh, "Performance analysis of pox and ryu with different sdn topologies," in *Proceedings of the 2018 International Conference on Information Science and System*, ser. ICISS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 244–249. [Online]. Available: https://doi.org/10.1145/3209914.3209931

[23] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 783–791.

[24] S. Haeri, Q. Ding, Z. Li, and L. Trajković, "Global resource capacity algorithm with path splitting for virtual network embedding," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 666–669.

[25] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1–9.

[26] H. Yu, V. Anand, C. Qiao, H. Di, and X. Wei, "A cost efficient design of virtual infrastructures with joint node and link mapping," *J. Netw. Syst. Manage.*, vol. 20, no. 1, p. 97–115, Mar. 2012. [Online]. Available: https://doi.org/10.1007/s10922-011-9209-x

[27] B. Dawes, D. Abraham, and R. Rivera. (2007) Fast network simulation setup (fnss) documentation. Accessed: 2020-03-06. [Online]. Available: https://www.boost.org/

[28] GNU. (2019) Gsl - gnu scientific library. Accessed: 2020-03-06. [Online]. Available: https://www.gnu.org/software/gsl/

[29] J. Nutaro. Adevs (a discrete event system simulator). Accessed: 2020-03-06. [Online]. Available: https://code.google.com/archive/p/hiberlite/

[30] GNU. (2012) Glpk (gnu linear programming kit). Accessed: 2020-03-06. [Online]. Available: https://www.gnu.org/software/glpk/

[31] Open-MPI. (2020) Open mpi: Open source high performance computing. Accessed: 2020-03-06. [Online]. Available: https://www.open-mpi.org/

[32] L. Saino, C. Cocora, and G. Pavlou. (2018) Fast network simulation setup (fnss) documentation. Accessed: 2020-03-06. [Online]. Available: https://fnss.readthedocs.io/en/latest/

[33] SQLite. Sqlite. Accessed: 2020-04-18. [Online]. Available: https://www.sqlite.org

[34] D. Browser. Db browser. Accessed: 2020-04-18. [Online]. Available: https://sqlitebrowser.org/

[35] Python. (2012) Python. Accessed: 2020-04-12. [Online]. Available: https://www.python.org/

[36] L. Saino, C. Cocora, and G. Pavlou. (2018) Fast network simulation setup (fnss) documentation. Accessed: 2020-03-06. [Online]. Available: https://fnss.github.io/

[37] N. Jahan, K. Bekshentayeva, S. Haeri, and A. Gonzalez. (2019) Vne-sim tool user guide. Accessed: 2020-03-06. [Online]. Available: http://www.sfu.ca/~ljilja/cnl/projects/VNE-Sim/vne-sim-web/VNE-Sim_Tool_User_Guide_updated.pdf

[38] K. Bilal, S. Khan, L. Zhang, H. Li, K. Hayat, S. Madani, N. Min Allah, L. Wang, D. Chen, m. i. Khan, C.-Z. Xu, and A. Zomaya, "Quantitative comparisons of the state-of-the-art data center architectures," *Concurrency and Computation: Practice and Experience*, vol. 25, 08 2013.